Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming

Leinonen, Antti

ACM
2019-02-22

Leinonen , A , Nygren , H , Pirttinen , N , Hellas , A & Leinonen , J 2019 , Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming . in Proceedings of the 50th ACM Technical Symposium on Computer Science Education . ACM , New York , pp. 84-90 , The 50th ACM Technical Symposium on Computer Science Education (SIGCSE 2019) , Minneapolis , Minnesota , United States , 27/02/2019 . https://doi.org/10.1145/3287324.3287378

http://hdl.handle.net/10138/315983
https://doi.org/10.1145/3287324.3287378

unspecified
acceptedVersion

Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming

Antti Leinonen
University of Helsinki
Helsinki, Finland
antti.leinonen@helsinki.fi

Henrik Nygren
University of Helsinki
Helsinki, Finland
henrik.nygren@helsinki.fi

Nea Pirttinen
University of Helsinki
Helsinki, Finland
nea.pirttinen@cs.helsinki.fi

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@cs.helsinki.fi

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

ABSTRACT

When learning programming, students learn the syntax of a programming language, the semantics underlying the syntax, and practice applying the language in solving programming problems. Research has suggested that simply the syntax may be hard to learn. In this article, we study difficulty of learning the syntax of a programming language. We have constructed a tool that provides students code that they write character-by-character. When writing, the tool automatically highlights each character in code that is incorrectly typed, and through the highlight-based feedback directs students into writing correct syntax. We conducted a randomized controlled trial in an introductory programming course organized in Java. One half of the population had the tool in the course material immediately before programming exercises where the practiced syntax was used, while the other half of the course population did not have the tool, thus approaching the exercises in a traditional way. Our results imply that isolated syntax writing practice may not be a meaningful addition to the arsenal used for teaching programming, at least when the programming course utilizes a large set of small programming exercises. We encourage researchers to replicate our work in contexts where syntax seems to be an issue.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Applied computing** → **Interactive learning environments**;

KEYWORDS

syntax practice, writing code, embedded tool

ACM Reference Format:
Antti Leinonen, Henrik Nygren, Nea Pirttinen, Arto Hellas, and Juho Leinonen. 2019. Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287378

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287378

1 INTRODUCTION

When learning to program, students work in an interlocked domain where challenges in one area contribute to challenges in other areas. For example, du Boulay [9] proposes five overlapping domains: understanding what programs are; forming an understanding of how computers execute programs; learning the notation and syntax of a language; learning structures that are used for solving programming problems; and learning the pragmatic skills needed for example in testing and debugging programs. When interviewing students, Lahtinen et al. [12] observed that difficulties with syntax are intertwined with other difficulties such as understanding program structures, understanding how to design programs, and dividing functionality of a program into smaller components.

Being able to write syntactically correct programs is a fundamental part of being able to program. Ng and Bereiter [16] suggest that learning to program starts with learning the syntax, which is followed by learning the structure and style. Here, making errors such as adding a semicolon after a conditional or a loop by mistake can take plenty of time to identify and fix [1]. Syntax errors are not problematic for only the struggling students as noted by Denny et al [8]: "*all students spent a similar amount of time solving the most common errors no matter what quartile they were in*".

Intuitively, if students practice writing syntax through mimicking small code samples, and the writing practice is designed so that the students receive character-by-character feedback on whether what they wrote is correct, students should be able to learn to avoid the major pitfalls related to typing the practiced syntactic constructs. Previously, the importance of training syntax has been highlighted in SyntaxTrain [15], which is a tool that shows students a syntax diagram – effectively a flow diagram – and information on possible errors on a line.

In order to study whether simple syntax writing practice reduces syntax errors and improves students' performance in related programming problems, we have constructed a tool that is used for practicing code writing. For each character that the student types in the tool, the tool shows whether the character corresponds to the expected input and consequently highlights errors in syntax.

Our tool differs from SyntaxTrain in three ways. First, there are no flow diagrams as the students have not yet learned how to interpret them, second, the feedback is given on a character-by-character level, and third, our tool is directly embedded within the material.

We have conducted a randomized controlled trial where a part of the course population was presented with the tool before programming exercises where the practiced syntax was first used, and the other part of the course population worked on the programming exercises without separate syntax practice. We study whether there are differences in the populations in terms of effort – when measured through programming exercise -specific events such as keystrokes within the programming environment – and time used for the programming exercises.

This article is organized as follows. The next section outlines existing research on errors that students encounter when writing programs and considers the meaningfulness of isolated practice. Then, in Section 3, we outline our methodology, which includes the research questions, the context of the study, and the tool. The results of the study are presented in Section 4, and the implications of the results are discussed in Section 5, which also includes limitations of our study. Finally, Section 6 concludes the article and outlines possible future work.

2 BACKGROUND

When learning to program, a big part of the issues that novice programmers face is related to syntax. Robins et al. studied problem distributions in a CS1 course [19]. They studied the different types of issues for which students seek help during their programming laboratory sessions. The majority of the issues that students faced were categorized as "trivial mechanics", which were defined as "trivial problems with little mechanical details". These included issues with braces, brackets, semicolons, typos and spelling, Java and file naming conventions, forgotten import statements, formatting output, tidiness, indenting, and comments. All of the trivial mechanics issues are syntax related.

Jadud [11] studied novice programmers' "compilation behavior", i.e. the way how novices behave when encountering a syntax error. Jadud observed that the efforts related to fixing syntax errors were related to course outcomes and that the capability of handling syntax errors – *Error Quotient* – could be used to predict exercise and exam grades. This signifies the importance of teaching students to tackle syntactic issues or avoiding them altogether by separating syntax practice from concept practice. His article reports also an interesting vignette that followed a student's behavior during a part of a compilation session: it might take over an hour to fix a simple misplacement of a beginning brace due to the student not being able to understand the error messages provided by the compiler.

The environment in which Jadud's work was conducted in, BlueJ [20], was later augmented with data collection facilities by Brown et al. [5]. In an analysis of 37 million compilation events from BlueJ, Altamdri et al. [1] identified a set of frequently observed syntax errors. These included:

(1) Unbalanced parentheses, brackets, braces or quotation marks, or trying to enclose one with another.
(2) Confusing an assignment operator with a comparison operator.
(3) Including parameter types in method calls.

When comparing the results of Altamdri et al. with other studies on compilation errors, there are discrepancies which may be related to the potpourri of backgrounds and teaching approaches. For example, in the work of Denny et al. [8], the most frequent syntax errors were "Cannot resolve identifier", type mismatch, and missing semicolon – none of which are highlighted as very frequent in the study by Altamdri et al.

Even seemingly very simple things like writing the print command in Java can cause problems to students. The way how students write their first programs was studied by Vihavainen et al. [25] who identified four distinct mistake categories related to writing the print command. These were as follows:

(1) mixing up upper- and lowercase letters, for example writing *System* in *System.out.println* with a lowercase *s*,
(2) using other characters instead of periods to separate words, for example writing *System-out-println*,
(3) various mistakes with string literals, such as forgetting quotation marks or the word *out* when trying to print something, and
(4) general typing mistakes without clear misconceptions.

One of the suggestions that Vihavainen et al. propose is that the root of many problems seems to lie within the syntax of programming languages, which is usually very different from natural languages. They also noted that modern programming environments provide students with plenty of support as they are writing their programs. They noticed that many of the students used copy and paste in the exercises during the first week, but most of them utilized the "sout" shortcut instead during the second week of the course. In addition, in their context, only a very small portion of the students submitted code with syntax errors. In the case of an error, students fixed it locally before submitting to the server.

As syntax seems to be an issue in programming, as evidenced by the preceding research, one could suggest practicing it. Many support dividing learned content into smaller pieces that are first practiced separately and then integrated together [14, 21–23, 26]. Such practice can be done, for example, under the guidance of a tutor or a peer who provides the learner with more challenging tasks as they are progressing, within a tutoring system, or within an e-book that interleaves theory and practice. Isolated practice can be beneficial for the learner as it can be used to reduce the cognitive load that is associated with the actual task [3].

As being able to write correct syntax is a cornerstone in writing programs, doing small syntax practice before programming tasks could be beneficial. For example, Ng and Bereiter [16] have suggested that learning the style and structure of programs are preceded by learning the syntax. Once students have internalized syntax, they should be able to ignore the details of the syntax during program design and construction, allowing them to direct attention to more relevant parts of the program [18].

3 RESEARCH DESIGN

Here, we first describe the context of the study and the syntax practice tool in more detail and then outline our research questions and methodology.

3.1 Context

The data for this study comes from a seven-week introductory programming course organized at the University of Helsinki during the fall of 2017. The programming course is the first course that freshmen who major in computer science take. Approximately one third of the course participants study computer science as their major, while the rest come from various backgrounds ranging from pedagogy to medicine.

The course follows an online textbook with theory, quiz, and programming exercise parts. The quizzes are done within the course material, while the programming exercises are completed in a separate programming environment that collects snapshots of students' programming process. The course also includes weekly lectures. The lectures are in the beginning of the week and the exercise deadlines at the end of the week. In this study, we focus on the first two weeks of the course, which cover the principles of procedural programming with Java. The topics include input/output, variables, conditionals, loops, and lists.

The course material has been written to follow the principles of the Extreme Apprenticeship method [26], which emphasizes that most of the students' time in the course should be spent on solving programming exercises. The majority of the exercises are small, and sequential programming exercises form larger programs as they are completed. The first two weeks under study have almost sixty programming exercises altogether.

3.2 Tool description

The tool consists of two main elements. The first element is used to show the code that the student has to write, and the second element is used to provide the input field into which the students are expected to write the code. The input field has three functionalities: first, the input field highlights wrongly written syntax, second, the input field blocks copying and pasting – that is, students must write the code –, and third, the input field records whatever the user writes for future analysis, given the online learning material provides a server for the tool where the data should be stored.

The syntax to highlight is generated from a template, which is also used to generate the code that the student must write. The code that the students are expected to write can be restricted, for example, to only a single statement. For example, in the example illustrated in Figure 1, the students do not need to write the class and method body. They are only expected to write the for-statement, i.e. the following:

```
for (int i = 0; i < 10; i++) {
    System.out.println(i * i);
}
```

Figure 1 also highlights what happens if the syntax is written incorrectly. In the example, the user has mistakenly added a semicolon after the `for`-loop statement. In the Figure, the highlight has been emphasized with an arrow for readability purposes. Once the code is correctly written, the red cross in the lower right corner of the input field changes into a green check mark.

The tool was embedded in the online learning material on the first and second week of the course. It was placed so that students would use it to practice the same syntactic keywords that were present in the following exercises. For example, the first iteration of the tool practiced Java's print statement, and it was placed right before the first programming exercises such as writing the traditional program that outputs "Hello World!".

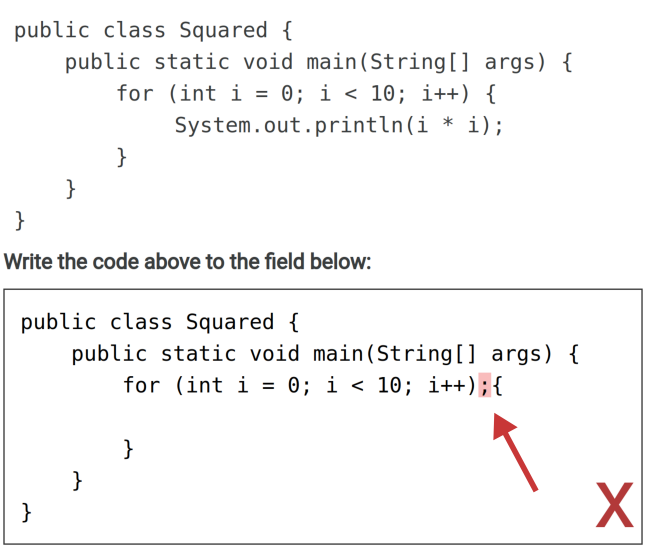

Figure 1: Illustration of the code that the student is expected to write and the input prompt of the tool. In the figure, the student has mistakenly added a semicolon after the for-statement, which is highlighted and must be fixed.

3.3 Data collection

Students encountered the tool in the material in the first two weeks for a total of five times. The tool was included when key syntax was introduced. The syntax in our study were the print statement (exercise 1), reading input (exercise 2), if-clause (exercise 3), while-clause (exercise 4), and method body (exercise 5), later referred to as Ex. 1 to 5. We examined data from the five instances where the tool was embedded in the material and each programming exercise that was given immediately after each syntax practice.

In order to determine the amount of time and the number of events required to complete the exercises, we used a NetBeans [4] programming environment plugin called Test My Code (TMC) [17], which collects students' keystrokes and events within the programming environment. The data also includes timestamps, which we used to study how much time students spent on the exercises.

For this study, we focused on character insertion and deletion events in order to analyze code written by the students themselves (and not for example copy-pasted). Similar insertion and deletion keystroke data with timestamps was collected from the tool.

3.4 Research questions

Our research questions for this study are as follows:

- RQ1. Do students who see the syntax practice tool have fewer events for the exercises when compared to those who did not see the tool?

- RQ2. Do students who see the syntax practice tool use less time for programming exercises when compared to those who did not see the tool?
- RQ3. Do students who see the syntax practice tool use less time in total for the syntax practice and programming exercises when compared to those who did not see the tool?

The study was conducted as a randomized controlled trial. The participants of the course were divided randomly into two groups A and B: group A (control) did not see the syntax practice tool in the material, while group B (treatment) did. In randomly assigned groups external factors such as initial skill levels should be approximately equal between the groups.

3.5 Group formation

As our study is an A/B-study, where group A consists of students who did not see the tool, and B of students who saw the tool, we compare these two groups in our analysis. Since the tool is voluntary to use and students in our study did not receive course points or other incentives for using the tool, some of the students in group B chose not to use the tool.

Thus, in addition to the A/B study, we further study group B (students who saw the tool) separately based on whether students in the group actually used the tool or not. In the analysis, the students from group B who did not use the tool were assigned into group C, while the students who used the tool were assigned into group D. Here, using the tool means that the student completed a syntax writing task in the tool correctly.

The division to groups C and D was done per exercise, depending on whether the student had completed the tool previous to the analyzed exercise. Exercise specific division was done so that the exercise data would better reflect the effects of the tool on the students' coding process for the said exercise as the completion of the tool later on in the material would not have helped on the earlier exercises. All participants had the same programming exercises.

3.6 Analysis

The purpose of the study was to analyze whether the syntax practice tool influences students' behavior in programming exercises. We study this both through the amount of individual events that students produce within the syntax practice tool for each particular syntax construct and within the programming environment in the programming exercises immediately following the syntax practices. In addition to the events, we also study the time that students used in the syntax practice tool and within the programming environment.

To answer the first research question, *Do students who see the syntax practice tool have fewer events for the exercises when compared to those who did not see the tool?*, we studied whether the students in group B had fewer keystroke events in the exercise data than the students in group A on average. We only included inserting and removing code as the number of times the students ran, tested or submitted their progress is irrelevant to the purpose of the tool.

To answer the second research question, *Do students who see the syntax practice tool use less time for programming exercises when compared to those who did not see the tool?*, we analyzed the time used for the exercise. We calculated the time between the first and last keystroke. We removed gaps in the snapshot timestamps that were over five minutes long as we did not want to include any pauses that students took. For this, we again compared the performance of group B against group A.

To answer the third research question, *Do students who see the syntax practice tool use less time in total for the syntax practice and programming exercises when compared to those who did not see the tool?*, we included the time spent on the tool for the students in group B when comparing time usage for the following exercise. This was done as it could be argued that even if students may complete exercises faster after using the tool, the tool is only useful if the total time in both the exercise and the tool is smaller for those students who saw the tool.

For all three comparisons, we used the Kolmogorov-Smirnov test [13] for examining whether the groups' differences were statistically significantly different, and corrected the results for multiple tests using the Holm-Sidak correction method [10].

4 RESULTS

4.1 Descriptive Statistics

A total of 249 students in the course consented for their data to be used in this research. From these 249 students, 17 had to be excluded as they did not provide the needed snapshot data. From the remaining 232 students, 125 students were in group A, meaning that they did not see the tool in the course material. Group B consisted of 107 students, which was split into groups C and D, that is, the students who saw the tool but ignored it (group C) and students who completed the tool (group D). The division to groups C and D was done per exercise, where group D consisted of the students who completed the tool previous to the analyzed exercise. Table 1a shows the number of students in groups A, B, C, and D per exercise.

4.2 Events and tool usage

Our first research question asks if the students who saw the tool had fewer typing events in the exercises after the tool than those who did not see the tool. Our hypothesis was that the students who saw the tool would have fewer events as they would make fewer typos while writing the syntax. However, it would seem that the tool did not help, that is, the students who saw the tool did not make fewer typos. Table 1b shows the median of the number of typing events between the groups. The differences between the number of typing events of the students who did not see the tool (group A) and those who saw it (group B) are not statistically significant based on the Kolmogorov-Smirnov test [13] in any of the exercises.

4.3 Time and tool usage

Our second research question focuses on whether the students who saw the tool used less time answering the programming exercises than those who did not see the tool. We studied the amount of time the students used on answering the programming exercises by calculating the time between their first and last keystrokes. We removed over five minute breaks between keystrokes to reduce the effect of taking breaks on the results. Table 1c shows the medians of the amount of time the group used per exercise. The differences in the time usage between groups are not statistically significant based on the Kolmogorov-Smirnov test.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5
No tool (A)	125	125	125	125	125
Tool (B)	107	107	107	107	107
Ignored tool (C)	50	36	50	47	59
Used tool (D)	57	71	57	60	48

(a) Number of students per group.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5
No tool (A)	52.68	141.56	146.96	164.69	60.74
Tool (B)	55.30	172.77	158.93	179.13	51.62
Ignored tool (C)	57.56	127.68	156.76	162.67	45.69
Used tool (D)	53.49	191.80	160.49	189.00	59.47

(b) Event counts (median) for completing the exercise following the syntax practice tool.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5
No tool (A)	1.90	3.80	2.91	3.99	0.84
Tool (B)	1.81	5.54	2.98	4.35	0.88
Ignored tool (C)	1.66	3.19	2.87	3.53	0.86
Used tool (D)	1.92	6.53	3.05	4.84	0.90

(c) Time in minutes (median) for completing the exercise following syntax practice tool.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5
No tool (A)	1.90	3.80	2.91	3.99	0.84
Tool (B)	2.19	6.22	4.65	5.07	1.51
Ignored tool (C)	1.73	3.68	3.55	3.76	1.11
Used tool (D)	2.56	7.29	5.43	5.86	2.05

(d) Time in minutes (median) when syntax practice and exercise time has been combined.

Table 1: Descriptive statistics of groups. Group A refers to students who did not have the tool in the material and B to students who did have the tool in the material. Group B is also split into two subgroups: Group C refers to those who saw the tool but did not use it and D to those who saw the tool and used it at least once. None of the differences between A and B are statistically significant based on the Kolmogorov-Smirnov test.

Our third research question is near identical to our second research question with the exception that we include the time used on the tool for the students who saw it. When comparing the time usages between the students who saw the tool (group B) and those who did not (group A), the results were not statistically significant based on the Kolmogorov-Smirnov test. The medians of time usage for groups A, B, C and D can be seen in table 1d.

Finally, we also compared the total time spent on exercises and the tools from groups C and D who both saw the tool. The comparison result from the Kolmogorov-Smirnov test was initially statistically significant showing difference between the groups, but after correcting our analysis for multiple comparisons using the Holm-Sidak correction method [10] the differences were not statistically significant.

5 DISCUSSION

5.1 Revisiting the research questions

In this study, we conducted a randomized controlled trial in order to determine whether light syntax practice would reduce the effort needed to complete related programming exercises in an introductory programming course. The effort that students invest into the exercises was studied from three perspectives: (1) the number of edit events in the exercises, (2) the amount of time that students spend on the exercise, and (3) the combined time in the practice system and the programming exercises.

The results, as outlined in the previous section, suggest that simple syntax practice does not improve students' performance in the subsequent programming exercises. No statistically significant differences were observed between the control group who did not have the syntax practice tool embedded to their learning material, and the treatment group who had the syntax practice tool immediately before the programming exercises where the syntax was used.

The results we observed suggest that the tool is not helpful, at least as it is. All of the results were statistically insignificant, which means that the people who saw the tool did not perform considerably better than those who did not see the tool. Next, we explore some of the possible explanations for our findings.

5.2 Separate syntax practice

The way how exercises are integrated into the learning environment plays a role in students' learning. If the exercises are placed in different locations or systems, it is possible that students must split their attention which increases extraneous cognitive load [6]. In our context, the syntax practice was conducted within the online learning material into which the syntax practice tool was embedded. The decision to embed the component into the learning material was partially driven by making a system that would be easy to share with others. In hindsight, however, it is possible that this decision also decreased the efficiency of the practice system.

From the perspective of reducing unnecessary cognitive load, supporting systems should be integrated into the environments in which students work [24]. For example, when programming exercises are worked on within a programming environment, having programming-related feedback such as syntax practice integrated into the environment would likely be a meaningful choice. On the other hand, if programming exercises are embedded to the online learning materials using tools such as Python Classroom Response System [28], it is possible that syntax practice within the online learning materials would be more beneficial as well.

Another way to address the issue, i.e. split-attention effect, was recently proposed by Altadmri et al [2]. They have developed a frame-based editing approach where syntactic commands, such as "if", once finished, are "glued" into place similar to block-based programming languages. While the system does not specifically focus on syntax practice, the approach shows promise in reducing problems with syntax.

In addition to the location of the syntax practice tool, another factor in its effectiveness could be its frequency in the course material. As the tool appeared only once per practiced syntactic keyword, simply having more syntax practice could have been beneficial.

5.3 Course organization

Course organization and management can influence students' learning. In the studied context, the course pedagogy focuses heavily on students working with small programming exercises that together combine into larger programs. The smallest exercises can effectively be seen as typing practice within the programming environment as the material provides examples that are very similar to the smallest exercises. It is possible that the use of small exercises within the course influences the need for syntax practice. There is some evidence that the use of such exercises can lead to students starting their work earlier, and thus also performing better [7].

We determined, in a post hoc fashion, whether some of the errors highlighted in related studies such as [1] were an issue in the studied context. We analyzed the keystroke data to identify conditionals and loops with functionality disabled with a semicolon, that is:

```
if (condition); {
    // code
}
while (condition); {
    // code
}
```

From the population of 232 students, none had encountered the above issues within the studied data set, which consisted of two weeks of an introductory programming course. While this is anecdotal and by no means representative of all syntax errors, it is possible that the course pedagogy and the way how the course is organized influences the errors that students encounter. Consequently, it could be meaningful to attempt to use our tool in contexts where students are more prone to struggle with syntax errors.

Furthermore, similar to the study reported in [25], many of the students used shortcuts when completing the course exercises. For example, some copied and pasted statements from the material and only changed the content, while others used shortcuts and autocomplete features. It is possible that the course format plays a role here too: the course has a two-hour weekly lecture where the teacher primarily focuses on live coding and worked examples. However, attendance is not mandatory and not all students attended the lecture. While we have no exact data on attendance, the responsible lecturer stated that approximately half of the students in the course attended the first two lectures relevant to this study.

5.4 Exercise complexity and syntax errors

Continuing with the above observation, that is, none of the students in the studied context had particular syntax errors, it is possible that the complexity of the studied programming exercises was too low. Due to the way the course is organized, the exercises that students are given after a new topic are initially small and then grow into larger problems. As syntax practice was conducted when a new topic was introduced, the exercises may have simply been too trivial for the students.

The complexity of problems and syntax errors have been linked in research. For example, as noted by Winslow [27],"*Given a new, unfamiliar language, the syntax is not the problem, learning how to use and combine the statements to achieve the desired effect is difficult*". The influence of the program complexity on syntax errors should be studied further in the future.

6 CONCLUSIONS

In this article, we reported a randomized controlled trial that evaluated the applicability of isolated syntax practice for learning programming. The isolated syntax practice was implemented as a component that was embedded in the learning materials immediately before the programming exercises where the students were expected to use the practiced syntax. The study was motivated through the research that points out that some struggle with syntax [1], and that the struggles with syntax errors are not only limited to the students who perform more poorly in courses [8].

Our results show that isolated syntax practice does not help student performance in subsequent programming exercises when measured in terms of total events or time needed to complete the exercises. Similarly, when comparing the total time used for the programming exercise combined with the typing practice, there was no statistically significant difference between the populations in the randomized controlled trial. At the same time, our experimental setup made it possible to not use the tool even if students were placed in the typing practice group.

In the discussion, we explored possibilities for why the result turned out as it did. A few of the possibilities include the pedagogy of the studied course: the participants in the course practice programming with tens of weekly exercises, some of which are used to highlight syntactic constructs that students are expected to learn. It is possible that the first exercises where syntax is practiced are sufficient and no separate practice is needed. It is also possible that the context in which the syntax practice was performed (that is, the online learning material) is too different from the programming environment where the programming exercises were worked on.

Currently, we are looking for other variables that could influence the outcomes such as previous programming background and other factors. Given, for example, information on previous programming background, we could look into whether the tool could be beneficial for students who have not previously programmed. We are also looking for researchers and educators whose students work with large programming exercises or whose students complete their programming exercises within online learning materials. Having someone replicate our study, with possibly contradictory results, would provide further insight into why syntax may be hard for some students.

REFERENCES

[1] Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 522–527.

[2] Amjad Altadmri, Michael Kolling, and Neil CC Brown. 2016. The cost of syntax and how to avoid it: Text versus frame-based editing. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 748–753.

[3] Paul Ayres. 2013. Can the isolated-elements strategy be improved by targeting points of high cognitive load for additional practice? *Learning and Instruction* 23 (2013), 115–124.

[4] Tim Boudreau, Jesse Glick, Simeon Greene, Vaughn Spurlin, and Jack J Woehr. 2002. *NetBeans: the definitive guide: developing, debugging, and deploying Java code.* " O'Reilly Media, Inc.".

[5] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: a large scale repository of novice programmers' activity. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 223–228.

[6] Paul Chandler and John Sweller. 1992. The split-attention effect as a factor in the design of instruction. *British Journal of Educational Psychology* 62, 2 (1992), 233–246.

[7] Paul Denny, Andrew Luxton-Reilly, Michelle Craig, and Andrew Petersen. 2018. Improving Complex Task Performance Using a Sequence of Simple Practice Tasks. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM, New York, NY, USA, 4–9. https://doi.org/10.1145/3197091.3197141

[8] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All Syntax Errors Are Not Equal. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 75–80. https://doi.org/10.1145/2325296.2325318

[9] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[10] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[11] Matthew C Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*. ACM, 73–84.

[12] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, NY, USA, 14–18. https://doi.org/10.1145/1067445.1067453

[13] Raul HC Lopes. 2011. Kolmogorov-smirnov test. In *International encyclopedia of statistical science*. Springer, 718–720.

[14] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2017. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 388–388. https://doi.org/10.1145/3059009.3081327

[15] Andreas Leon Aagaard Moth, Joergen Villadsen, and Mordechai Ben-Ari. 2011. SyntaxTrain: relieving the pain of learning syntax. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 387–387.

[16] Evelyn Ng and Carl Bereiter. 1991. Three levels of goal orientation in learning. *Journal of the Learning Sciences* 1, 3-4 (1991), 243–271.

[17] Martin Pärtel, Matti Luukkainen, Arto Vihavainen, and Thomas Vikberg. 2013. Test my code. *International Journal of Technology Enhanced Learning 2* 5, 3-4 (2013), 271–283.

[18] Robert S Rist. 1989. Schema creation in programming. *Cognitive Science* 13, 3 (1989), 389–414.

[19] Anthony Robins, Patricia Haden, and Sandy Garner. 2006. Problem distributions in a CS1 course. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., 165–173.

[20] Dean Sanders, Phillip Heeler, and Carol Spradling. 2001. Introduction to BlueJ: a Java development environment. In *Journal of Computing Sciences in Colleges*, Vol. 16. Consortium for Computing Sciences in Colleges, 257–258.

[21] R Keith Sawyer. 2005. *The Cambridge handbook of the learning sciences*. Cambridge University Press.

[22] Juha Sorva and Otto Seppälä. 2014. Research-based Design of the First Weeks of CS1. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research (Koli Calling '14)*. ACM, New York, NY, USA, 71–80. https://doi.org/10.1145/2674683.2674690

[23] Jeroen JG Van Merriënboer, Richard E Clark, and Marcel BM De Croock. 2002. Blueprints for complex learning: The 4C/ID-model. *Educational technology research and development* 50, 2 (2002), 39–61.

[24] Jeroen JG Van Merriënboer, Paul A Kirschner, and Liesbeth Kester. 2003. Taking the load off a learner's mind: Instructional design for complex learning. *Educational psychologist* 38, 1 (2003), 5–13.

[25] Arto Vihavainen, Juha Helminen, and Petri Ihantola. 2014. How Novices Tackle Their First Lines of Code in an IDE: Analysis of Programming Session Traces. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research (Koli Calling '14)*. ACM, New York, NY, USA, 109–116. https://doi.org/10.1145/2674683.2674692

[26] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. 2011. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 93–98. https://doi.org/10.1145/1953163.1953196

[27] Leon E. Winslow. 1996. Programming Pedagogy&Mdash;a Psychological Overview. *SIGCSE Bull.* 28, 3 (Sept. 1996), 17–22. https://doi.org/10.1145/234867.234872

[28] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. 2013. Facilitating code-writing in PI classes. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 585–590.